Robot Motion 2

✡Coordinate Frames 2
✡Degree of Motion/Movement Freedom 2
✡Holonomic Ground robot 2
✡Differential Drive 2
✡Actuation of driving wheels–DC Motors 3
✡Motor Control 3
✡Mapping wheel rotation speed to velocity 4
✡Position-Based Path Planning 4
Practical: locomotion, calibration and accurate motion 5
✡Uncertainty in Motion 5

Sensors 5

✡Proprioceptive and Outward-looking 5
✡Single and Multiple Value Sensors 6
✡Servoing 7
✡Probabilistic Sensor Modelling 8
Practical: Wall Following with Sonar 8

Probabilistic Robotics 8

✡Probabilistic Localization 8
✡Bayesian Probabilistic Inference 9
✡Steps in Monte Carlo Localization/Particle Filter (Continuous Localization) 9
✡Compass Sensor 11

Place Recognition and Occupancy Mapping 11

✡Place Recognition(Global Localization via Place Recognition) 11
✡Probabilistic Occupancy Grid Mapping 12

Simultaneous Localization and Mapping (SLAM) and Planning13

✡SLAM 13
✡Local Planning: Dynamic Window Approach 13
✡Global Planning: Wavefront Method & Rapidly Exploring Randomized Trees (RRT) 13

Robotics integrates science and engineering, and overlaps with many disciplines (artificial intelligence; computer vision, perception; machine learning, estimation, inference; neuroscience; electronics, mechanical engineering).

Robot Motion

✡Locomotion: a mobile robot can move and sense, and must process information to link these two.

goals/design points of robot movement/locomotion system:
- speed and/or acceleration of movement
- precision of positioning (repeatability = do the same thing)
- flexibility and robustness in different conditions
- efficiency, low power consumption (possibly)

Robots can move in water, in the air, on land, in space. Example: AUV, Micro UAV, Zero-G Assistant (0 gravity), spider, humanoid

✡Coordinate Frames

2 coordinate system: world frame W anchored/fixed in the world;
robot frame R carried by and stays fixed relative to the robot(rigid body) at all time

Knowing the robot location = transformation between frames W and R

✡Degree of Motion/Movement Freedom

A rigid body which translates and rotates along a 1D path has 1 DOF: translational (train)
A rigid body which translates and rotates on a 2D plane has 3 DOF: 2 translational, 1 rotational (ground robot)
A rigid body which translates and rotates in a 3D volume has 6 DOF: 3 translational, 3 rotational (flying robot)

✡Holonomic Ground robot

A holonomic robot is one which is able to move instantaneously in any direction in the space of its DOF.
- a train is holonomic, car is not holonomic (it cannot slide sideways)
- car is under-actuated, 2 controlled inputs (forward speed, steering angle, not enough to give full 3DOF of general movements)

Holonomic robots do exist, but need many motors or unusual designs and are often impractical.
Ground-based holonomic robots can be made using omnidirectional wheels.
- over-actuated, 4 inputs of different speed to cover 3 DOFs.

exotic wheeled robots
- Segway RMP(dynamic balance)
  if robot tips, control by acceleration to adjust the center of mass
- Mars Rover(rocker-bogie, wheel + legged design)
- self-balancing lego
  measure the reflected brightness OR use gyro sensor to measure the tilting

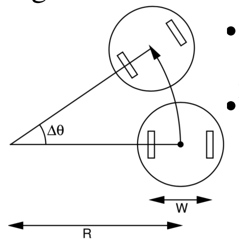
standard wheel—non-holonomic (a car-like robot cannot instantaneously move sideways)

✡Differential Drive

two motors, one per wheel/rack, steering achieved by setting different speeds
a (or two) caster wheel/smooth ball to balance the robot with little friction
wheels run at equal speeds for straight-line motion
wheels run at equal and opposite speeds to turn on the spot
combinations of speeds lead to motion in a circular arc (steer)
$v = r\omega$ linear velocity versus angular velocity

Circular Path of a differential drive robot

To find radius R of curved path, consider a period of motion Δt where the robot moves along a circular arc through angle Δθ



- left wheel: distance moved=$v_L \Delta t$
- radius of arc=$R - \frac{W}{2}$ right wheel: distance moved=$v_R \Delta t$ radius of arc=$R + \frac{W}{2}$

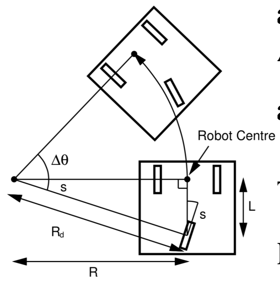
- both wheel arcs subtend the same angle Δθ, so $\Delta\theta = \frac{v_L \Delta t}{R - \frac{W}{2}} = \frac{v_R \Delta t}{R + \frac{W}{2}}$ and $\frac{W}{2}(v_R + v_L) = R(v_R - v_L)$
- $R = \frac{W(v_R + v_L)}{2(v_R - v_L)}$ $\Delta\theta = \frac{(v_R - v_L)\Delta t}{W}$ $v_R = V_L$, $R \to \infty$, straight forward; $v_R = -v_L$, $R \to 0$, rotate on the spot

Odometry: understand how robot moves (in space) on the command given (velocities of wheels)

Car/Tricycle/Rack and Pinion Drive

two motors, one to drive, one to steer
cannot normally turn on the spot
with a fixed speed and steering angle, it will follow a circular path
with four wheels, need rear differential and variable(Ackerman) linkage for steering wheels
wheels can drive different speeds, but average speed remains the same

Circular Path of a Car-Like Tricycle Robot



a single steerable and drivable wheel at the back. The front wheels are free running
Assume no sideways wheel slip, we intersect the axes of the front and back wheels to form a right-angle triangle and obtain $R = \frac{L}{\tan s}$

The radius of path that the rear driving wheel moves in is $R_d = \frac{L}{\sin s}$

In time Δt, the distance along its circular arc moved by the drive wheel is vΔt, so the angle Δθ through which the robot rotates is $\Delta\theta = \frac{v\Delta t}{R_d} = \frac{v\Delta t \sin s}{L}$ $R = \frac{L}{\tan s}$ $\Delta\theta = \frac{v\Delta t \sin s}{L}$

✡Actuation of driving wheels—DC Motors

DC motors are available in all sizes and types
Simple control with voltage or Pulse Width Modulation
For precision, encoders and feed back can be used for servo control.
standard DC motors tend to offer high speed and low torque, so gearing is always required to drive a robot
Gearing: high speed, low torque → low speed, high torque (hard to stall)

- If gear 1 is driven with torque t1, it exerts tangential force $F = t_1 / r_1$ Force × Radius = Torque
- On gear 2, the torque is $F = r_2 F = (r_2 / r_1) t_1$ (boosted by the ratio)
- The change in angular velocity between Gear 1 and Gear 2 is calculated by considering velocity at the point where they meet $v = \omega_1 r_1 = \omega_2 r_2$ so $\omega_2 = (r_1 / r_2)\omega_1$ (reduced by the same ratio)
- When a small gear drives a bigger gear, the second gear has higher torque and lower angular velocity in proportion to the ratio of teeth
- Gears can be chained together to achieve compound effects (series of gears)

encoder: optical sensor + black&white dot plate for feedback control

✡Motor Control

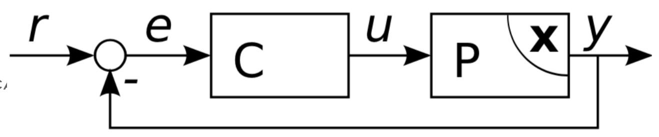
1.Open Loop

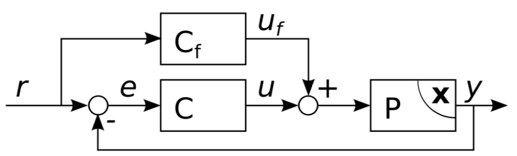
- Let P be a single-input-single-output dynamic system(Lego Mindstorms geared DC motor)
- input u—voltage V or corresponding PWM value
- internal states x, whose dynamics follow differential equations—x=[ω, ϕ] with rotation speed ω and angle ϕ
- output y as a function of x—angle ϕ, y=x2

2.Closed Loop

- Let C be a controller, r is a reference/desired output, e is error between reference and actual output
- The controller runs at a high frequency. At each iteration, it checks the current error and calculates a control value to the motor which aims to reduce the error
- Simplest controller—binary ON/OFF (thermostat). If the error is positive, go forward; If it is negative, go back

3.PID Proportional-Integral-Differential control





- controller $u(t)=k_p e(t)+k_i\int_{t_0}^{t} e(\tau)d\tau+k_d\frac{de(t)}{dt}$ (e(t) error always positive)
- proportional gain (reduce the error)
  kp small, sluggish behaviour, long time until converge; kp large, overshoot & oscillation
- integral gain (remove steady-state error)
- differential gain (reduce settling time and remove oscillating behaviour)
- Heuristic Tuning rule—Ziegler-Nichols
  set ki and kd to zero, increase kp until system starts oscillating with period Pu (in seconds)
  remember this gain as ku. set kp=0.6ku, ki=2kp/Pu, kd=kp*Pu/8

4.Additional Tweaks

reference filtering—respect physical limits already in the reference
Anti-Reset-Windup—stop integrating the error for the I-part, when u is at its physical limit
Dead-band Compensation—add offset to u to compensate friction
Feed-forward controller C$_f$— $u_f(t)=k_f\frac{dr(t)}{dt}$ to reduce "work" for the feedback controller

✡Mapping wheel rotation speed to velocity

In principle, we could measure the radius of each wheel r$_w$ to turn angular velocity into linear motion as $v=r_w\omega$ (error-prone).
In practice (due to hard to model factors, such as surface slip and tyre softness) it is better to calibrate such things empirically. i.e. via experiments (guided trial and error), work out the scaling between the motor reference angle and distance travelled over the ground.

1.Motion and State on a 2D plane

Assume a bout is confined to move on a plane, its location can be defined with a state vector x $\bar{x}=\begin{pmatrix}x\\y\\\theta\end{pmatrix}$

x, y specify the location of the pre-defined robot center point in the world frame
θ specifies the rotation angle between the two coordinate frames (the angle between xW and xR axes)
The two coordinate frame coincide when the robot is at the origin and x=y=θ=0

2.2D motion on a plane—3 DOF represented by (x, y, θ) with -π< θ≤ π

Consider a robot which only drives ahead or turns on the spot

During a straight-line period of motion of distance D $\begin{pmatrix}x_{new}\\y_{new}\\\theta_{new}\end{pmatrix}=\begin{pmatrix}x+D\cos\theta\\y+D\sin\theta\\\theta\end{pmatrix}$

During a pure rotation of angle α $\begin{pmatrix}x_{new}\\y_{new}\\\theta_{new}\end{pmatrix}=\begin{pmatrix}x\\y\\\theta+\alpha\end{pmatrix}$

3.Integrating circular motion estimates in 2D (Kinematics)

For differential drive and tricycle robot, for periods of constant circular motion

$$\begin{pmatrix}x_{new}\\y_{new}\\\theta_{new}\end{pmatrix}=\begin{pmatrix}x+R\left[\sin(\theta+\Delta\theta)-\sin\theta\right]\\y-R\left[\cos(\theta+\Delta\theta)-\cos\theta\right]\\\theta+\Delta\theta\end{pmatrix} \quad \text{where } R=\frac{W(v_R+v_L)}{2(v_R-v_L)} \quad \Delta\theta=\frac{(v_R-v_L)\Delta t}{W}$$

✡Position-Based Path Planning

Assuming that a robot has localization, and knows where it is relative to a fixed coordinate frame, then position-based path planning enables it to move in a precise way along a sequence of pre-defined waypoints. Paths of various curved shapes could be planned, aiming to optimize criteria such as overall time or power usage. Here we will consider the specific, simple case where we assume that:

Our robot's movements are composed by straight-line segments separated by turns on the spot.

The robot aims to minimize total distance travelled, so it always turns immediately to face the next waypoint and drives straight towards it.

Assume that the robot's current pose is (x, y, θ) and the next waypoint to travel to is at (W_x,W_y)

- The vector direction it must point in is $\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} W_x - x \\ W_y - y \end{pmatrix}$
- It first rotate to point towards the waypoint, $\alpha = \tan^{-1}\left(d_y / d_x\right)$
- Care must be taken to make sure that α is in the correct quadrant of $-\pi < \alpha \le \pi$
- A standard $\tan^{-1}$ function will return a value in the range $-\pi/2 < \alpha \le \pi/2$
- The angle the robot must rotate through is $\beta=\alpha-\theta$. If the robot is to move as efficiently as possible, care should be taken to shift this angle by adding or subtracting 2π so make sure that $-\pi < \beta \le \pi$.
- The robot should then drive forward in a straight line through distance $d = \sqrt{d_x^2 + d_y^2}$

Practical: locomotion, calibration and accurate motion

gradual motion drift from perfect square

causes: initial alignment errors, wheel slip, mis-calibration, unequal left/right motors

Careful tuning of PID control and calibration of distance and angle improves matters but we will never achieve a perfect result every time in this experiment.

A well-calibrated(systematic error removed) robot

After careful calibration the robot should on average return to the desired location, but scatter will remain due to uncontrollable factors (variable wheel slip, rough surface, air currents)

Systematic error removed (via calibration), what remains are zero mean errors.

The errors occur incrementally: every small additional movement or rotation induces a little more potential error

The size of the distribution of the errors in the world frame will grow as the robot moves further around the square

Model the zero mean errors probabilistically by a Gaussian normal distribution

Perfect motion integration from odometry is not possible.

✡Uncertainty in Motion

'ideal' trajectory is affected by uncertain perturbations(motion noise).

During a straight-line period of motion of distance D $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x+(D+e)\cos\theta \\ y+(D+e)\sin\theta \\ \theta + f \end{pmatrix}$

During a pure rotation of angle α $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$

e, g, f are uncertainty terms, with zero mean and a Gaussian distribution, which model how actual motion might deviate from the ideal trajectory

Adding these terms won't help us to move a robot more accurately when it is guided with only odometry; but are important later when we probabilistically combine odometry with other sensing.

Sensors

Sensors gather numerical readings or measurements.

✡Proprioceptive and Outward-looking

1.**Proprioceptive**(self-sensing) sensors will improve a robot's sense of its own internal state and motion.

e.g. motor encoders, internal force sensors

- the value of measurement z_p will depend on (a function of) the <u>state of the robot x</u>: $z_p=z_p(x)$

- the state of a robot is a vector of variables used to describe its current status

  for a simple robot moving on a flat ground plane x=(x, y, θ)
- proprioceptive measurement might depend not just on the current state but also previous states in the robot's history or the current rate of change of state

  wheel odometry will report a reading depending on the difference between the current and previous state

  gyro, a part of an Inertial Measurement Unit (IMU), reports a reading depending on the current rate of rotation

2.But without **outward-looking** sensors a mobile robot is moving blindly.

- localize without drift with respect to a map
- recognize places and objects it has seen before
- map out free space and avoid obstacles
- interact with objects and people
- In general, be aware of its environment

A measurement from an outward-looking sensor will depend both on the state of the robot x and the state of the world around it y: $z_o=z_o(x, y)$

The state of world might be parameterized in many ways—a list of geometric coordinates of walls/landmarks (may either be uncertain or perfectly known)

✡Single and Multiple Value Sensors

1.Touch, light, sonar sensors return a single value within a given range.

①Touch Sensor returns yes/no state, to detect collisions and trigger avoidance action (negative feedback)

- Binary On/Off state—no processing required
- Switch Open—on current flows
- Switch closed—current flows (hit)

②Light Sensor

- detect intensity of light incident from a single forward direction with some range of angular sensitivity
- multiple sensors pointing in different directions can guide steering behaviors
- The Lego sensors also have a mode where they emit their own light, which will reflect off close targets and can be used for following a line on the floor or quite effective short-range obstacle avoidance.
- passive mode—sense incoming light (obstacle avoidance)
- active mode—emit light and sense reflected lights (proximity, infrared sensor, black and white sensor)

③Sonar/Ultrasonic Sensor

returns depth value in cm, to smooth servoing behaviour with proportional gain(negative feedback)

- measures depth/distance by emitting an ultrasonic pulse and timing the interval until echo returns
- sonar beam typically has an angular width of 10~20 degrees
- fairly accurate depth measurement (centimetre) in one direction, but can give noisy measurements in the presence of complicated shapes (20cm, 30cm ~ 1m, 2m)
- maximum range a few metres
- robots sometimes have a ring of sonar sensors for obstacle detection
- Especially important underwater(almost 0 light) where it is the only serious option beyond very short ranges.

2.Sensors, such as a camera or laser range-finder return an array of (numerical) values.

It can be achieved by scanning a single sensing element as in a laser range-finder, OR, by having an array of sensing elements, such as the pixels of a camera's CCD chip

External Sensing: Laser Range-Finder

External Sensing: Vision (passive)

camera is cheap and small, better than LRF. If can understand the picture.

angle of light —> lenses—> position on sensor (but no shape/geometry info)

Touch Sensors for Bump Detection

Multiple Touch Sensor (active 1 or 2 touch sensors)

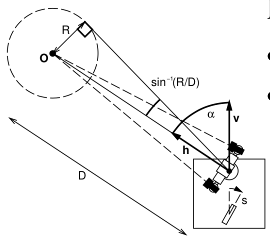
✡Servoing

Servoing is a robot control technique where control parameters(such as desired speed of a motor) are coupled directly to a sensor reading and updated regularly in a negative feedback loop (closed loop control).
Servoing needs high frequency update of the sensor/control cycle or motion may oscillate.
The PID control we implemented last week using the encoders in a motor is one example of servoing and negative feedback. This concept can also be used with outward-looking sensors.

1.Proportional Control using an External Sensor: Servoing
In servoing, a control demand is set which over time aims to bring the current value of a sensor reading into agreement with a desired value.
Proportional control: set demand proportional to negative error (difference between desired sensor value and actual sensor value): set velocity proportional to error $v = -k_p\left(z_{desired} - z_{actual}\right)$
where k_p is the proportional gain constant.
(Note that since this is a different control loop, k_p will not have the same value as in last week's motor tuning but will need to be individually adjusted through trial and error)

Outward-Looking Sensor Servoing
- In the previous slide we saw a proportional law to control velocity; but in our robots when we set a velocity demand there is another level of PID control underneath (as we looked at last week) which monitors the motor encoders translates velocity demands into a sequence of PWM values which are actually sent to the motors.
- This is an example of cascade control where the output of one control loop is used to set the input to another. While in principle both controllers could be combined into a unified whole, this is a very useful abstraction which hides the details of motor/encoder control from the upper level work with sensor control. Problems will not arise unless the top level controller requests velocity changes too rapidly, exceeding the bandwidth of the lower level controller.
- Outward looking sensors such as sonar or cameras will sometimes produce 'garbage' readings for a number of different possible reasons. It can often be sensible to use a strategy to try to remove these outliers, such as median filtering which doesn't use the most recent sensor value in the control law but a smoothed value such as the median of the past n measurements. Note though that this will also reduce the responsiveness of the system.

2.Visual Servoing to Control Steering



For a robot with a tricycle or car-type wheel configuration, s is steering angle
- simple steering law which will guide robot to collide with target: $s = k_p\alpha$
- steering law which will guide robot to avoid obstacle at a safe radius: subtract offset
$s = k_p\left(\alpha - \sin^{-1} R / D\right)$
where R is the radius of the obstacle and D is the distance from robot center to obstacle center

Combining: sensing/action loops
No modelling and planning! Consider each local 'servo'-like sensing-action loop as a behaviour.
Sense $\rightarrow$ Act
The challenge is in combining many behaviours into useful overall activity: TR Programs, Subsumption, Braitenberg vehicles.

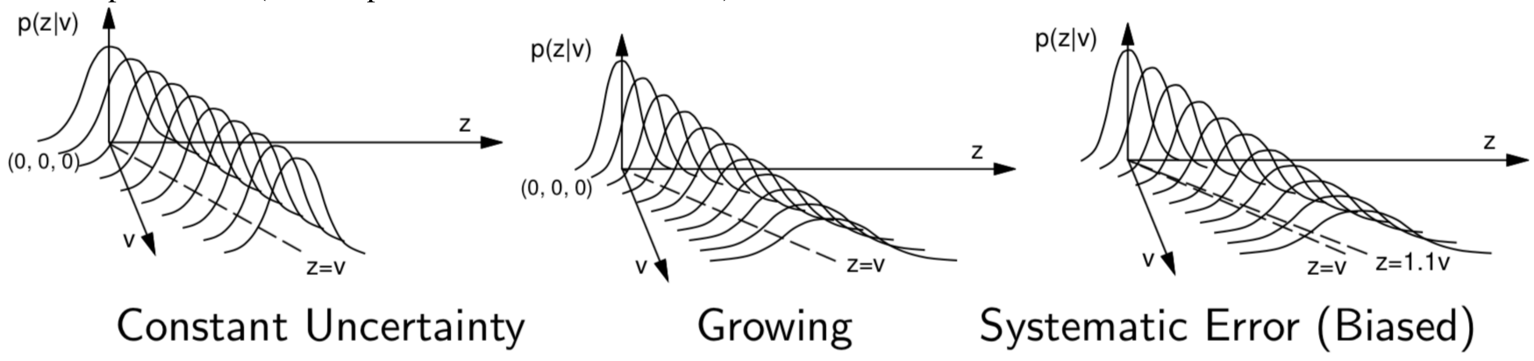
Combining Sensors: World Model Approach
- Capture data; store and manipulate it using symbolic representations.
- Plan a sequence of actions to achieve a given goal
- Execute plan.
- If the world changes during execution, stop and re-plan.
- Powerful, but computationally expensive and complicated!
- Probabilistic state inference and planning is the modern version of this able to cope with uncertainty in sensors.

✡Probabilistic Sensor Modelling

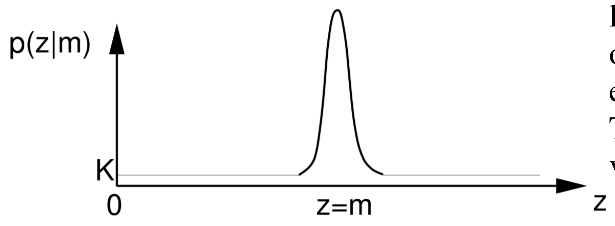
Like robot motion, robot sensing is also fundamentally uncertain. Real sensors do not report the exact truth of the quantities they are measuring but a perturbed version.

A probabilistic measurement model for how sensor works—a probability distribution (specifically a likelihood function) of the form: $p(z_o \mid x, y)$ with a Gaussian shape.

A likelihood function fully describes a sensor's performance.
$p(z|v)$ is a function of both measurement variables z and ground truth v and can be plotted as a **probability surface**.
For a depth sensor: (surface plot of the whole likelihood)



Robust Likelihood for Sonar Measurements



likelihood function for a sonar sensor—what is the probability of obtaining sensor measurement z given that the ground truth value I expect is m?
This distribution has a <u>narrow Gaussian band</u> around the expected value, plus a <u>constant additive band</u> representing a fixed percentage of 'garbage' measurements $P(z|m) \propto e^{-\frac{(z-m)^2}{2\sigma_s^2}} + K$

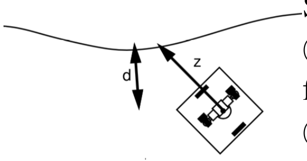
Practical: Wall Following with Sonar

Use **sideways-looking sonar** to measure distance z to wall
Use velocity control and a while loop at for instance 20Hz
With the goal of maintaining a desired distance d, set difference between left and right wheel velocities proportional to difference between z and d: $v_R - v_L = k_p(z-d)$

Symmetric behaviour can therefore be achieved using a constant offset v_c: $v_R = v_C + \frac{1}{2}k_p(z-d) \quad v_L = v_C - \frac{1}{2}k_p(z-d)$

where vc is the speed of robot driving forward even if not parallel to the wall
when z>d, turn forward and inward a little

Problem: If angle between robot's direction and wall gets too large, because <u>sonar doesn't measure the perpendicular distance</u>.



Solutions:
①ring of sonar sensors would be most straightforward. Clever combination of measurements from different times, and take the small values.
②A simple way to a better is to mount the sonar forward of the wheels. This couples rotation and the distance from the wall

Probabilistic Robotics

✡Probabilistic Localization

To know where the robot is

sequential Probabilistic filter—Kalman filter
Particle filter—based on samples

✡Bayesian Probabilistic Inference

'Bayesian' has come to be known as a certain view of the meaning of probability theory as a measure of subjective belief.
Probabilities describe our state of knowledge — nothing to do with randomness in the world.
Sensor Fusion: the general process of combining data from many different sources into useful estimates.

Particles: hypothesis of a robot position.
weighted average of particles = estimate of robot location

Preliminaries for MCL:
—update a probability distribution using particles, to represent motion uncertainty.
—want to see the particles spreading out in a realistic manner to represent uncertainty over a square trajectory
—Waypoint-based Navigation
—Investigate the performance of the sonar sensor probabilistically.

Two ways of thinking about how MCL works:
- A Bayesian probabilistic filter.
- 'Survival of the fittest', like a genetic algorithm. After motion prediction, the particles are in a set of random positions which should span the possible positions into which the robot might have really moved. When a measurement is made from sonar, the likelihood each particle is assigned, according to how well it fits the measurement, is a 'fitness' score. Normalizing and resampling then allow strong particles to reproduce (multiple copies are made of them), while weak particles die out.
  good hypothesis survive (increase weights), bad die (decrease/delete).

MCL can be used to tackle both of these problems, the only difference is in how we initialize the particle set.
- **Continuous Localization** is a tracking problem: given a good estimate of where the robot was at the last time-step and some new measurements, estimate its new position.
  - In continuous localization we usually assume that we start from a perfectly known robot position: set the state of all particles to the same value. Set all weights to be equal. The result is a 'spike', completely certain point estimate of the robot's location. $x_1 = x_2 = \ldots = x_N = x_{init}$; $w_1 = w_2 = \ldots w_N = 1/N$
- **Global Localization**('kidnapped robot problem'): the environment is known, but the robot's position within it is completely uncertain. (blind motion at the beginning)
  - In global localization we start from knowledge only that the robot is somewhere within a certain region. The state of each particle should be sampled randomly from all of the possible positions within that region. Set all weights to be equal. x_i = Random ; $w_1 = w_2 = \ldots w_N = 1/N$

✡Steps in Monte Carlo Localization/Particle Filter (Continuous Localization)

These steps are repeated every time the robot moves a little and makes measurements:
In MCL, a <u>cloud of weighted particles</u> represents the uncertain position of a robot.

①Motion Prediction based on Odometry

- During a straight-line period of motion of distance D $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + (D+e)\cos\theta \\ y + (D+e)\sin\theta \\ \theta + f \end{pmatrix}$

  when forwarding, the sideway spreading is small e, f ~2cm, 3cm
- During a pure rotation of angle α $\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$

  e, f and g are zero mean noise terms — i.e. random numbers sampled from with a Gaussian distribution
  variance (e^2, f^2, g^2) should vary linearly with distance or angle, not standard deviation
  scale the variance to moving distance var ∝ distance σ ∝√(distance)

Note: make sure that θ values are always in the range −π to π.

Solve this by simply adding or subtracting 2π to any particles which go out of this range.

②Measurement Update based on Outward Looking Sensors (Sonar)

- $P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}$ and $w_{i(new)} = P(z|x_i) \times w_i$
  - P($z|x_i$) is the likelihood of particle i — probability of getting measurement z given that x_i represents the true state
  - Note the denominator in Bayes' rule P(z) is a constant factor, independent of the position of robot, which we do not need to calculate because it will later be removed by normalization.
- If robot is at pose (x, y, θ) then its forward distance to an infinite wall passing through (Ax , Ay) and (Bx , By)
  - Get ground Truth Value $m = \frac{(B_y - A_y)(A_x - x) - (B_x - A_x)(A_y - y)}{(B_y - A_y)\cos\theta - (B_x - A_x)\sin\theta}$
  - The world coordinates at which the forward vector from the robot will meet the wall are $\begin{pmatrix} x + m\cos\theta \\ y + m\cos\theta \end{pmatrix}$

    check if the sonar should hit between the endpoint limits of the wall
- If the sonar would independently hit several of these walls, obviously the closest is the one it will actually respond to.
- If the angle between the sonar direction and the normal to the wall is too great, probably the sonar will not give a sensible reading and you should ignore it. $\beta = \cos^{-1}\left(\frac{\cos\theta(A_y - B_y) + \sin\theta(B_x - A_x)}{\sqrt{(A_y - B_y)^2 + (B_x - A_x)^2}} \right)$
- Likelihood for Sonar Update
  - The likelihood should depend on the difference (z−m). If this is small, then the measurement validates a particle; if it is big, it weakens the particle.
  - The further away the measurement is from the prediction, the less likely it is to occur. A Gaussian function is usually a good model for the mathematical form of this. The standard deviation σ_s is based on our model of how uncertain the sensor is, and may depend on z or may be constant. $P(z|m) \propto e^{-\frac{(z-m)^2}{2\sigma_s^2}}$ $\sigma_s \approx$2–3cm **$(\sigma_s)^2 \propto$ depth**
  - Note the difference between this and the motion prediction step. There we sampled randomly from a Gaussian distribution to move each particle by a slightly different amount. Here in the measurement update we just read off a value from a Gaussian function to obtain a likelihood for each particle.
  - A robust likelihood function models the fact that real sensors sometimes report 'garbage' values which are not close to ground truth. Robust functions have 'heavy tails'. This can be achieved most simply by adding a constant to the likelihood function. The meaning of this is that there is some constant probability that the sensor will return a garbage value, uniformly distributed across the range of the sensor. $P(z|m) \propto e^{-\frac{(z-m)^2}{2\sigma_s^2}} + K$

    add tail (constant offset) → wont kill the particles that far from the predictions, maybe 10% tail
  - The effect of a robust likelihood function in MCL is that the filter is less aggressive in 'killing off' particles which are far from agreeing with measurements. An occasional garbage measurement will not lead to the sudden death of all of the particles in good positions.

③Normalization

The weights of all particles should be scaled so that they again add up to 1.

Calculate ∑wi and divide all existing weights by $w_{i(new)} = \frac{w_i}{\sum_{i=1}^{N} w_i}$

④Resampling

- Resampling consists of generating a new set of N particles which all have equal weights 1/N, but whose spatial distribution now reflects the probability density.
- To generate each of the N new particles, we copy the state of one of the previous set of particles with probability according to that particle's weight.
- This is best achieved by generating the cumulative probability distribution of the particles, generating a random number between 0 and 1 and then picking the particle whose cumulative probability this intersects.
- For efficiency it is possible to skip the normalization step and resample directly from an unnormalized distribution.

✡Compass Sensor

A digital compass gives a robot the ability to estimate its rotation without drift.
Having a digital compass and a single sonar sensor puts us in a position close to that of having a full sonar ring.
In principle, the robot could stop and rotate on the spot every so often to simulate a sonar ring.
We could simply monitor the compass as the robot continuously moves and feed this into our filter.

magnet compass sensor reports NSWE bearings (measure the orientation of robot)

Compass measures bearing β relative to north. What is $P(\beta|x_i)$?

- Clearly the likelihood only depends on the θ part of x_i .
- If the compass is working perfectly, then $\beta=\gamma-\theta$, where γ is the magnetic bearing of the x coordinate axis of frame W.
- So we should assess the uncertainty in the compass, and set a likelihood which depends on the difference between β and $\gamma-\theta$, Gaussian $e^{-\frac{(\beta-(\gamma-\theta))^2}{2\sigma_c^2}}$
- Particles far from the right orientation will get low weights.

Integrate compass measurement and have another update on particles after updates from sonars.

Place Recognition and Occupancy Mapping

Global Localization/Kidnapped Robot with sonar
One Depth Measurement and Resampling
Using a sonar and compass together (can reduce ambiguity)

✡Place Recognition(Global Localization via Place Recognition)

An alternative relocalization technique involves making a lot of measurements at a number of chosen locations and learning their characteristics.
This can be done without a prior map but needs training.
The robot can only recognize the locations it has learned.
For instance: at each location, spin the robot and take a regularly spaced set of sonar measurements (e.g. one per degree).

- First the robot must be placed in each target location to learn its appearance.
- The raw measurements are stored to describe the location: **a place descriptor/signature** (pre-learned)
  e.g [robot angle vs depth measurement] histogram
- Afterwards, the robot is placed in one of the locations and it must take a set of measurements then decide which it is in
- It must see which saved signature matches best to the new measurements by checking each in turn.
- Two histograms can be compared with a correlation test, measuring the sum of squared differences.
  Difference D_k between new measurement histogram $H_m(i)$ and saved signature histogram $H_k(i)$ is

$$D_k = \sum_i \left(H_m(i) - H_k(i)\right)^2$$ (sum of square of difference of each individual bar)

- The saved location with lowest D_k is the **most likely candidate**,
- Also check that D_k is below a **threshold**, in case the robot is in none of the known locations.

- If the test histogram and that from one of the saved locations can be brought into close agreement by only a shift, the robot is in the same place but rotated.
- The amount of shift to get the best agreement is a measurement of the rotation.

Depth Measurement Histogram [depth vs frequency measurement] histogram

- To save the computational cost of always trying every shift, we can build a signature which is invariant to robot rotation, such as a histogram of occurrences of certain depth measurements.
- Matching tests can then be carried out on this directly.
- Once the correct location has been found, the shifting procedure to find the robot's orientation need only be carried out for that one location.

✡Probabilistic Occupancy Grid Mapping

A probabilistic algorithm for mapping in the case that a robot's localization is known.
The goal is to infer which parts of the environment around a robot are navigable free-space, and which parts contain obstacles.
In Occupancy Mapping, rather than building a parametric map of the positions of walls we use a regular grid representation.
Occupancy grids accumulate the uncertain information from sensors like sonar to solidify towards precise maps.

state of robot = (x, y, θ), orientation of sonar=$\theta+\alpha$ with respect to standard coordinate frame
①Test whether the cell inside the beam

vector from robot to the cell C_i, $C_i = g_i - r = \begin{bmatrix} g_x - x \\ g_y - y \end{bmatrix}$ where gi is global frame is cell i

compare C_i to z, $z = \begin{bmatrix} z\cos(\theta+\alpha) \\ z\sin(\theta+\alpha) \end{bmatrix}$ where z is the vector of sonar facing

β: angle between measurement vector z and vector from robot to the cell C_i $\beta = \arccos \frac{C_i \cdot z_i}{\|C_i\| \cdot \|z_i\|}$

β could be +ve or -ve $|\beta|<H$, where $H \approx 5°$ (beam width)
②distance—compare length of C_i and z
if $-\sigma < \|z\| - \|C_i\| < +\sigma$, increase probability
if $\|z\| - \|C_i\| > \sigma$, decrease probability
where σ is uncertainties of sonar (standard deviation)
Else, not change probability (not learn any info about those cells)

You need to loop over all of the cells in the grid, and for each one calculate the vector c from the robot to that cell. You then compare this with the measurement vector z from the robot to the measured point. You look at the angle between z and c (found by computing the dot product); if that is less than the half beam width then this is a cell of interest. Then you compare the length of vectors z and c. If their difference is less than the standard deviation then this is a cell at the end of the beam whose occupancy log odds should be increased; if z-c is greater than sigma then this is a cell within the free space part of the beam whose log odds should be decreased. The amounts to increase or decrease by can be chosen fairly freely in this question; e.g. increase by 5, decrease by 2.

$P(O_i) + P(E_i) = 1$
Initialize the occupancy probabilities for unexplored space to a constant prior value (for instance 0.5)

$P(O_i|Z) = \frac{P(Z|O_i)P(O_i)}{P(Z)}$ $P(E_i|Z) = \frac{P(Z|E_i)P(E_i)}{P(Z)}$

log-odds $\frac{P(O_i|Z)}{P(E_i|Z)} = \frac{P(Z|O_i)}{P(Z|E_i)} \frac{P(O_i)}{P(E_i)}$ $o(A) = \frac{P(A)}{P(\bar{A})}$ $o(O_i|Z) = \frac{P(Z|O_i)}{P(Z|E_i)} * o(O_i)$

Taking logs: $\ln o(O_i|Z) = \ln \frac{P(Z|O_i)}{P(Z|E_i)} + \ln o(O_i)$

For each cell we store $\ln o(O_i)$ and update it additively.
Cells with probability 0.5 of occupancy will have log odds 0; positive log odds means probability > 0.5; negative log odds means probability < 0.5. Also we normally *cap* log odds at certain positive and negative limits.

We need models for the likelihood function which models sensor performance. We can consider directly the ratio of likelihoods we need to update log odds.
Log odds update $U = \ln \frac{P(Z|O_i)}{P(Z|E_i)}$: for each cell P(Z|Oi) is the probability of obtaining the sensor value given that the cell is occupied; P(Z|Ei) is the probability of obtaining that value given that the cell is empty.

- For cells within the sonar beam but closer than the measured depth Z=d, P(Z|Oi)/P(Z|Ei) is less than 1; we can choose a constant negative value for U
- For cells within the sonar at around the measured depth Z=d, P(Z|Oi)/P(Z|Ei) is greater than 1; we can choose a constant positive value for U.

Note that we are oversimplifying in occupancy grids in assuming the probabilities of occupancy for each cell are independent.

Simultaneous Localization and Mapping (SLAM) and Planning

✡SLAM

✡Local Planning: Dynamic Window Approach

If a robot had a map of environment, it wants to make a plan to get from one place to another.
To plan a path around a complicated set of obstacles (adaptive path of movements)
—Consider robot dynamics and possible changes in motion it can make within small time dt.
—For each possible motion look ahead longer time τ. Calculate benefit/cost based on distance from target and obstacles.

- Robot can control vL, vR up to maximum values.
- Max acceleration × dt is the maximum change we can make to in time dt (e.g 0.1s).
- 9 possible actions/trajectories at each step—each of vL, vR can either go down, up or stay the same.
- Predict the new position after longer time τ (e.g 1s) for each action.
  robot moves on a circular arc
- For each motion calculate benefit and cost
  Benefit will be one weight times the amount the robot would moved closer to its target at (Tx, Ty)
  $$B = W_B \times D_F = W_B \times \left[\sqrt{(T_x - x)^2 + (T_y - y)^2} - \sqrt{(T_x - x_{new})^2 + (T_y - y_{new})^2} \right]$$
  W_B: to be tuned for a good performance
  Subtract from this a cost which is based on the distance the robot would be from the closest obstacle at (Ox, Oy)
  $$C = W_C \times \left[D_{safe} - \left(\sqrt{(O_x - x_{new})^2 + (O_y - y_{new})^2} - r_{robot} - r_{obstacle} \right) \right]$$ as penalty
  Find (Ox, Oy) by search through obstacles. D_{safe} is some distance we would ideally like to stay from contact.
  r_{robot} and $r_{obstacle}$ are robot and obstacle radii. (C should > 0, if C<0, set C=0)
- Choose the path with the maximum B–C and execute for dt.

—Choose the best and execute for dt, then do it again.

✡Global Planning: Wavefront Method & Rapidly Exploring Randomized Trees (RRT)

Wavefront Method
—divide map into grids, expand from finish or start point
Brute force ‘flood fill’ breadth first search of whole environment.
Guaranteed to find shortest route, but slow and computational expensive.

Rapidly Exploring Randomized Trees (RRT) Method
—faster than wavefront, no grid, build graphs, random chose points, connect the points to the closest node, finally get a path through open regions
Algorithm grows a tree of connected nodes by randomly sampling points and extending the tree a short step from the closest node.
Expands rapidly into new areas, but without the same guarantees.